Tcl Minimal Escaping Style

A **minimal Tcl escaping style** is a code style that eschews unnecessary Tcl escaping syntax.

See also

A question of style
A discussion about when to brace expressions. The Tcl minimal escaping style has nothing to say about that particular scenario, and certainly does not discourage proper escaping of expressions to avoid double substitution.

You only need curly braces...
Looking from a different angle.

Description

In some languages such as C, Javascript, Lisp, and Python, double quotes indicate a certain type of value: a string. In Tcl all values are already strings, so double quotes do not have that effect. Instead, they provide an environment in which whitespace characters are ordinary characters rather than word delimiters. Braces do the same, but additionally inhibit the standard substitutions. It's technically possible to use backslash substitution to accomplish the same thing that quotes or braces do for all values except the empty string, but in all non-trivial cases this results in unreadable scripts.

Some Tcl scripts use double quotes even when not strictly necessary. Sometimes this is done to make a text editor highlight those values, to make Tcl code look a little more like C or a Unix shell, or to indicate that a value is data rather than code to be evaluated by the interpreter. Someone learning Tcl might get from this style the wrong idea that double quotes indicate strings and that braces indicate lists. The interpreter may also have to do a little extra work to look for substitutions.

The minimal Tcl escaping style presented here makes a script more readable by using only escape characters only as strictly necessary to achieve the desired effect. Here are the rules:

braces
Don't use braces where no backslash substitutions are otherwise needed.

quotes
Don't use quotes where braces could be used.

That's it. Code that follows this style is both more readable and more performant. It also works in a greater variety of scenarios such as code generation. Beginners reading such scripts will get a better sense of Tcl, understanding the real intent of braces and quotes, and perhaps avoid the all-to-common misunderstanding that braces mean "list".

The format of a canonical list, which does not use double quotes since no variable or command substitutions are done, conforms to this minimal escaping style.

MS dons his evil hat and remarks: in that case, *really* minimal style could be without quotes or braces, just backslash-escape the whitespaces! Of course, it depends on how you define *minimal* ...

PYK 2015-04-30: Except that braces and quotes were presumably introduced into the language specifically to alleviate backslash pain. This minimal escaping style merely advocates using them only for their respective intended purposes. Note that the minimal escaping style says nothing about backslash escaping, which is always fair game.

PYK 2022-04-06: Also, there is one word that can't be expressed in a script without either quotes or braces: The empty string.

In the minimal escaping style, quotes are used for a value that contains both whitespace and substitutions , since braces would over-escape the value, preventing the substitutions from occurring.

Some routines such as expr, regexp interpret their arguments accoding to their own domain-specific language. The domain-specific language of expr happens to be so similar to Tcl proper that it could in some ways be considered a superset of Tcl. The Tcl minimal quoting style can be used in the context of expr as well, as long as literal non-numeric values are quoted. To conform to the minimal escaping syntax, the command,

```
expr {$mystring eq "hello, world"}
```

would instead be:

```
expr {$mystring eq {hello, world}}
```

PYK 2015-08-31: Over at C-like structs and file scope, Brian Griffin has this to say, and I think it also applies to some of the arguments critical of the minimal escaping style:

IMHO, using C syntax, even a little bit, is a bad idea. The more Tcl looks like C, the more you raise the expectation of C style parsing as well. This is the one area that screws with programmers minds the most. Because Tcl looks similar to C, the false conclusion is reached that it's parsed like C, which couldn't be farther from the truth. Tcl is just not C, so stop trying to turn it into C. Any programmer worth their salt can be multilingual; you don't need to baby them with fake syntax.

Examples

```
#instead of
puts "Hello!"

#use
puts Hello!

#instead of
puts "Hello World."

#use
puts {Hello, World.}

#instead of
proc {x} { ... }

#use
proc x { ... }
```

Counterpoint

A few reasons to use a less grammar-bound style of quoting:

- Syntax highlighting and "type" identification. Writing strings quoted with "" generally causes editors to highlight them differently, and conveys to the reader that the value is intended to be treated as a string. The visual cues make code easier to read on the wiki, in an editor or printed on paper.
- It's arguably more common to make a change like "Hello, world!" -> "Hello, $whom!". Not having to change delimiters makes such changes easier and less bug-prone.
- proc foo {a} {...} is another case where leaving out the first set of {} just makes later edits more cumbersome.

These points can be summarised in the famous Abelson and Sussman quote: "Programs must be written for people to read, and only incidentally for machines to execute." Optimise for the human reader, not the mechanical parser.

aspect can see the minimal escaping style as a useful pedagogical device for users who haven't yet internalised the dodekalogue, but finds its use elsewhere rather bizarre. Changing existing examples to this style without specific justification for that example seems gratuitous and potentially harmful.

This might be a good place to collect other examples of divergent coding styles .. Heronian triangles uses some short definitions to make equations work with minimal punctuation, at the cost of performance and safety.

PYK 2015-05-17: Wouldn't the proper solution here be to fix your editor's Tcl syntax highlighting algorithms, rather than to change your coding style just for the sake of an editor that "doesn't get it?" The problem with the "convey to the reader that the value is

intended to be treated as a string" idea is that in Tcl **every value is intended to be treated as a string**. I don't think this "type identification" idea holds any water at all. Rather, it obfuscates the nature of Tcl. If I ever looked at a piece of Tcl code and thought, "oh, this value is enclosed in quotes -- now I know it's going to be used as a string", it was back before I really grokked Tcl. Instead, to understand what's happening in any given command, I look at the command name, the value itself, and its position. I'd love to see some examples where double quotes actually add to a Tcl script the kind of value you're talking about. I suspect that any such examples that are brought up would be just as easily distinguished as strings if they were enclosed in braces, and that those who prefer quotes are just carrying habits over from other languages they work in. I agree that the minimal style isn't strictly adhered to nearly as widely as it should be, and that as more Tclers become more vigilant about it, there will be a corresponding increase in the speed at which newcomers grasp the nuances of the rules of Tcl.

Discussion

RLE: PYK your 'rules' are starting to stretch and crack under the strain here... To gain a starting point, and so no one has to keep scrolling back, here are your rules as they stand today (2015-05-19):

Here are the rules of the minimal escaping style:

braces
Don't use braces where no backslash substitutions are otherwise needed.

quotes
Don't use quotes where braces could be used.

That's it. ...

So, first, you are now starting to talk about "In the normal Tcl context" and "in the expr context". But your rules, as you've set them out above, contain absolutely no mention of any 'context' sensitive aspect at all. So, no language in any of your rules mentioning context means there is no contextual limitation to your rules as you've enumerated them so far. So your now adding a hidden contextual limitation is a big red flag that your rules are straining to stand up under load.

So, lets look at the rules, number one, expanding the contraction, says "do not use braces where no backslash substitutions are otherwise needed". Ignoring the double negative for the moment, this says that when there is "no need for backslash substitutions" one should "not use braces".

Ok, so, for "expr {$mystring eq hello}, since there are zero backslash subs in this command, there is "no need for backslash substitutions" and rule one says "do not use braces". Therefore writing "expr {$mystring eq {hello}} would "use braces" when rule one

says "do not use braces", and thereby violate rule 1. So to meet rule 1, we can not use braces.

So, on to rule two: "do not use quotes where braces could be used". Since to follow rule one, one "can not use braces", then in the expr, "braces can not be used" as required by rule 1. So since we ca not use braces in the expr, rule two says that quotes should, in fact, be used. I.e., since we can't use braces, due to rule one, then rule 2 does not apply, and so, [expr {$mystring eq "hello"}] is actually within what your rules proscribe (at least as you have written them, with the double negative assertion in rule 1).

So, what we have here is a nice contradiction. You say your rules require "expr {$mystring {hello}}" and your rules themselves, when parsed, say that the proper, minimal, quoting is "expr {$mystring "hello"}". Two different answers, from the same set of rules.

PYK 2015-05-20: As we all know, the rules of Tcl take a minimal approach, and this minimalism yields much more than initially meets the eye. Its prime offering is the ability to design little languages with ease. When it came time to tack on expr, we ended up with a little language very similar to Tcl itself, but with a few extra constraints. Barewords were made off-limits in order to save some room for expansion of the expr grammar at a later date. expr happens to be the most visible of little languages of Tcl, but it's not part of Tcl proper, and its syntax is not covered in the Tcl rules. The minimal escaping rules don't aspire to cover all the Tcl-like little languages that might spring up. On the one hand, it's asking a lot to demand that a set of rules cover two different grammars (not even the Tcl rules manage that), but on the other hand, I think the minimal escaping style can easily extend to expr syntax. All that's needed is to exclude rule 1. The rules are written such that they proscribe rather than prescribe braces and quotes given certain conditions, so one approach is simply to understand that because of the additional bareword constraint in expr, rule 1 doesn't apply to its syntax. I think that the "no backslash substitutions are otherwise needed" is already perfectly sufficient for that, as that part makes no sense for expr grammar. RLE's interpretation can be summarized as, "Doc, it hurts when I appy this rule to a grammar where the rule is nonsensical." The obvious answer is, **Don't do that!** If we wanted to be really explicit we could simply add, "... or they are needed to avoid barewords in expr expressions," and we're golden again. I really don't see the need for that, though. After all, The Tcl rules also leave many things implicit, such as the grammar of list.

PYK 2018-07-01: Additionally, in the expr example above, backslash rules would otherwise be needed in order to avoid double substitution, so it's actually an example of the goodness of rule #1.

EMJ (2015-04-22) Re "Sometimes this is done..." near the beginning: it might be done to clarify the purpose of a string, or with an eye on a future change (and so is about program maintainability). And I just don't believe the "seem more like C" bit.

PYK 2015-04-30: Anyone who thinks they can improve this page is free to dive in and rewrite it wholesale as far as I'm concerned....

EMJ (2015-04-22) No they can't, you'll put their name on the rewrites and try to turn them into another bit of your own private but global edit war!

PYK 2015-04-30: I added your name to your comments because they were in discussion format, expressing a different opinion. In that case, the speakers need to be labeled so the reader can understand that it's different people talking. If you don't want to attribute yourself, you could always defer to *anonymous coward* or something. In any case, other readers already aren't going to understand what we're discussing here because the page has already taken on another form. That's cool. It'll just need some cleanup later.

RS 2015-05-13: Some exceptions:

> semicolons must be quoted/braced if not used as statement separators:

```
set x [split $y ";"]
```

> small characters are better visible if redundantly quoted/braced: e.g. "."

PYK 2015-05-13: To conform to the minimal escaping style, that would either be

```
set x [split $y \;]
```

or

```
set x [split $y {;}]
```

The idea behind the minimal escaping style is specifically to not use braces quotes or backslash for side purposes such as increasing visibility of characters, appeasing syntax highlighters, or posing as another language. To be justified under the minimal escaping style, a brace, quote, or backslash must have a programmatic effect. Says me :)

MiHa 2015-05-18: So, what is the point of replacing "xx" with {xx} ? onforming to some random style is no value per se. {xx} doesn't get syntax-highlighted as a string by the wiki. I count that as a disadvantage.

If some style makes it harder for the user to see / understand / work / modify the code, is bad.

Is there some "official" / regular style ?

RLE 2015-05-18: There is no "official" or regular style, per se., as in "imposed by others". There are some suggestions that have the advantage of making code easier to read (i.e., indenting one's nested blocks), but nothing required/official, other than that required in order to get the Tcl parser to parse and execute the script.

For what it is worth, the "point" of replacing "xx" with {xx} is essentially arbitrary and decided by PYK. It is simply yet another version of the great space-comma rampage of last year by PYK, only at least with this rampage PYK has a more logical and reasonable

reason for the changes. But in the end, it is simply PYK imposing his "one world view" on everyone else yet again (albeit with a more reasoned and reasonable excuse behind the imposition).

EMJ (2015-05-18) No sensible syntax-highlighter for Tcl is going to highlight {xx} as text because then most of the code would be highlighted as text.

PYK 2015-05-18: It's not arbitrary, but informed by the rules of Tcl. In the case of "xx", the double quotes have no functional value since they aren't escaping any whitespace. For that matter, the same is true of {xx}. To conform to the minimal escaping style, it would simply be xx. As an example, the first random page I came across just now that uses double quotes is dictutils. They only occur in a couple of spots, and here is one of them:

```
foreachLine l myfile.tcl { puts [format "%-4d | %s" [incr count] $l] }
```

The minimal escaping style for this line would be:

```
foreachLine l myfile.tcl { puts [format {%-4d | %s} [incr count] $l] }
```

More difficult to read? Nah, but certainly more informative to newcomers who can note that from a functional standpoint only braces are necessary to get the job done. The main point of the Tcl minimal escaping style is readability, actually, since its wide adoption would help reduce quoting hell issues among newcomers -- in my experience the primary source of the most unreadable and contorted Tcl code out there.

When I started using the wiki a couple of years ago, the vast majority of its code examples weren't highlighted at all. I've put many hours into fixing that (dkf has as well), so I guess syntax-highlighted examples on the wiki are yet another one of my rampages imposing my "one world view" on everyone :)

EMJ 2015-05-18: No, it's an example of the perfectly reasonable wiki-gnome activity of converting a page to use a useful but previously-unavailable piece of wiki-formatting without any actual change in content (assuming you manage to not accidentally change the content).

BTW,

```
foreachLine l myfile.tcl { puts [format {%-${nwidth}d | %s} [incr count] $l] }
```

I can see a reason for doing the above, but I seem to have forgotten that now I **have** to use double quotes!

PYK: Eat more casein protein!

EMJ 2015-05-18: It would be nice (or perhaps not) if I could see any point in that remark.

PYK: The point is that quotes are not there to save the programmer from forgetting to use the right escaping mechanism for their needs in a given scenario. That's more a function of size of dessert portions, caffeine levels, domestic felicity, etc. We all make typos, forget

to link variables from other scopes, and commit any number of other programming mistakes. The purpose of double quotes is not to save the programmer from braces, but to permit substitutions **when necessary**. If double quotes are used where braces would have sufficed, another more important signal to the reader is sacrificed, namely that **"There's (almost) no substitution going on here, folks. Move along."**

EMJ 2015-05-18: You claim that your minimal quoting helps beginners. I don't agree. Even if it did, for a short while, you are actually hampering their ability to eventually understand how the language works, and to be able to understand code not written in this style. My normal approach is to use double- quotes unless I **need** to suppress substitution, so when I see later that I have used braces other than as an **apparent** part of the language structure, I know that there was a reason for it. Not so different from your argument really, just the other way round. But I can still cope with other styles, it may just take a little longer sometimes. As for all this diet etc. stuff, we know that how you feel affects your mistake rate, but it's got nothing to do with a choice of programming style.

PYK: An assertion like "hampering their ability to understand code not written in this style" deserves some evidence or explanation. Have you got any? I believe quite the opposite, that it enhances their ability to understand code written in any style. This "apparent part of the language structure" stuff is exactly what messes newcomers up. It's so easy to start off with that mistaken conception about braces, and hold it for a long, long time. Just ask scoofy in the Tcl Chatroom.

JMS - 2023-11-04 23:54:53

Based on some of the points in the discussion, I think "the rules" could be clarified a bit to use clearer wording and remove ambiguity. I propose the following four-clause algorithm:

1. If a string (value, command argument, etc) can be represented unambiguously as is, without any quoting or escaping, then it shall be represented as is.
2. Otherwise, if a string can be escaped by simply surrounding it with braces, then it shall be surrounded by braces.
3. Otherwise, if the string contains spaces *(or semicolons?)*, it shall be surrounded by double-quotes, with all special characters backslash-escaped. (This looks cleaner and is often shorter than escaping every individual space, like the list command does when it can't use braces.)
4. Otherwise, the string shall be written without surrounding braces or quotes, with all special characters backslash-escaped.

(Note: this happens to be exactly *how I'd define list's canonical representation if given the chance, rather than its current, kinda inconsistent and not quite formally defined behavior.)*

However, I wouldn't say this style should *always* be followed with no flexibility at all; in some cases it may be justified to bend the rules a bit for the sake of program readability (and leaving "hints" to future developers of your code), for example when it may be

unclear whether something needs quoting, or to highlight the fact that a bareword is actually a list or a command/script (e.g., you may prefer to write «proc foo args return» as «proc foo {args} {return}»), or simply when a different choice of representation would be clearly more concise or more readable.

---

dusthillresident - Tue 7 Nov 11:37:29 GMT 2023

I agree very much with the Counterpoint, it says pretty much everything I've been thinking while reading about this style.

I think the examples of the style on this page only show how the style is more harm than help. As soon as I saw them, I instantly thought:

1. "But what if later on you want to add more to that message? Then you end up adding quotes or braces anyway..."

2. "But what if later on you want to make it say 'Good (morning/afternoon/evening) (username), the current time is (current time)", then you have to change the braces to quotes anyway..."

3. "But then later on you have to add braces there anyway if it later turns out that your procedure needs to take more arguments..."

It reminds me of stuff I've done in the past when I've been thinking like "let's be clever and write something as short and compact as possible!", only to find myself having to fix and remove my "cleverness" later on when expanding the code, either because I was adding new features or for other reasons.

I understand the dodekalogue well, but I still use quotes or braces or write other potentially technically sub-optimal constructs deliberately, as a way to express intent, very much like how I might use phrasing carefully when using human language. Subtle details can sometimes express a lot and make a big difference to clarity.

Syntax highlighting in text editors is another valid reason not to adopt this style. The suggestion given in response to this, "fix your text editor", mentioned further up on this page, is really not practical.

I come away from this unsure of the value of this style, I see more reasons to avoid it than to adopt it. At least from the impression I'm getting from this page, it feels more like an obsessive purism kind of thing rather than something that has a sound logical or rational basis.

I can imagine though that this style could have some merit, in some particular situations I can imagine it improve code clarity as well as performance. I think better examples are needed though, the current examples as I said really don't show how the style could be of any benefit at all.
